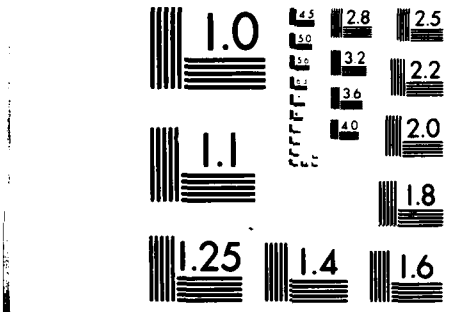
AD-A112 240 AERONAUTICAL RESEARCH LABS MELBOURNE (AUSTRALIA) F/G 9/2
HYBRID COMPUTING SYSTEM MARK 3 SOFTWARE REFERENCE MANUAL.(U)
JAN 81 H A THELANDER
UNCLASSIFIED ARL/SYS-NOTE-76 NL

END
DATE FILMED
4 82
DTIC



(12)

ARL-SYS-NOTE-76 AR-002-253

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

AERONAUTICAL RESEARCH LABORATORIES

MELBOURNE, VICTORIA

SYSTEMS NOTE 76

HYBRID COMPUTING SYSTEM MARK 3 SOFTWARE REFERENCE MANUAL

by

H. A. THELANDER

Approved for Public Release

COPY No 013

JANUARY 1981

(12)

AR-002-253

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
AERONAUTICAL RESEARCH LABORATORIES

SYSTEMS NOTE 76

HYBRID COMPUTING SYSTEM MARK 3 SOFTWARE REFERENCE MANUAL

by

H. A. THELANDER

SUMMARY

Hybrid Computing System Mark 3 (HCS3) comprises the A.R.L. DECsystem-10 central timesharing digital computer and the Modular Analogue Computer. In HCS3, digital programs are prepared in FORTRAN and use calls to subprograms in the system software package H3PAC to perform hybrid computations. This report forms the reference manual for the HCS3 digital applications programmer. It describes the calling and argument conventions, and the functions, of the subprograms in H3PAC. It also describes the associated PDP-11/20 disk utility package RKPAC, used after some HCS3 operations for recovery of data logged during computation.

POSTAL ADDRESS: Chief Superintendent, Aeronautical Research Laboratories, Box 4331, P.O., Melbourne, Victoria, 3001, Australia.

DOCUMENT CONTROL DATA SHEET

Security classification of this page: Unclassified

1. Document Numbers
   (a) AR Number: AR-002-253
   (b) Document Series and Number: Systems Note 76
   (c) Report Number: ARL-Sys-Note-76

2. Security Classification
   (a) Complete document: Unclassified
   (b) Title in isolation: Unclassified
   (c) Summary in isolation: Unclassified

3. Title: HYBRID COMPUTING SYSTEM MARK 3 SOFTWARE REFERENCE MANUAL

4. Personal Author: Thelander, H. A.

5. Document Date: January, 1981

6. Type of Report and Period Covered:

7. Corporate Author(s): Aeronautical Research Laboratories

8. Reference Numbers
   (a) Task: DST 20/22
   (b) Sponsoring Agency:

9. Cost Code: 71 6160

10. Imprint: Aeronautical Research Laboratories, Melbourne

11. Computer Program(s) (Title(s) and language(s)):
    H3PAC MACRO-10
    CISHCS MACRO-11

12. Release Limitations (of the document): Approved for public release

12.0. Overseas:	N.O.		P.R.	1	A		B		C		D		E	

13. Announcement Limitations (of the information on this page): No Limitation

14. Descriptors:
    Hybrid computers
    Programming manuals
    Computer systems programs
    Computer programming

15. Cosati Codes:
    0902
    0502

16. *ABSTRACT*

Hybrid Computing System Mark 3 (HCS3) comprises the A.R.L. DECsystem-10 central timesharing digital computer and the Modular Analogue Computer. In HCS3, digital programs are prepared in FORTRAN and use calls to subprograms in the system software package H3PAC to perform hybrid computations. This report forms the reference manual for the HCS3 digital applications programmer. It describes the calling and argument conventions, and the functions, of the subprograms in H3PAC. It also describes the associated PDP-11/20 disk utility package RKPAC, used after some HCS3 operations for recovery of data logged during computation.

CONTENTS

1. INTRODUCTION

This manual provides the programmer's reference to the System Software for HCS3, the Aeronautical Research Laboratories Hybrid Computing System Mark 3. In HCS3, the analogue part of the computations is carried out by the Modular Analogue Computer (MAC), while the digital part is done by the Aeronautical Research Laboratories' DECsystem-10 central time-sharing computer. The hybrid interfacing, data logging, and high-speed digital function generation for hybrid computation, together with Command and Information System (CIS) functions, are performed by the PDP-11/20.

1.1 Software Packages

The HCS3 system software is based upon two software packages. The first is H3PAC, written in the MACRO-10 assembler language, which contains subprograms which are called from a user program written in the FORTRAN IV (F10) language. Calls to this package enable the user programs to perform hybrid computations.

The second package is CISHCS, written in MACRO-11 assembler, which runs in the PDP-11/20 and supervises the transfer of data and control between the MAC and DECsystem-10. CISHCS also executes functions in support of the CIS display and its DECsystem-10 package CISPAC, and supports the RKPAC PDP-11/20 RK05 disk utility package, described in Chapter 16 below.

1.2 Hardware Configuration

The hardware configuration which supports HCS3 is shown in Figure 1. Digital computations are performed by the DECsystem-10, analogue computations by the MAC, and hybrid interfacing is performed by the PDP-11/20. Peripherals of the PDP-11/20 include 24K words of core, an Extended Arithmetic Element (EAE), a KW11-P programmable real time clock, the CIS controller, two RK05 disk cartridge drives, and the Digital Serial Bus (DIGIBUS) which performs various digital-type operations inside the MAC.

The MAC has ten racks, each with a capacity for forty operational amplifiers, of which ten may be used as integrators. Up to ten transfer function modules may be employed in each rack, utilising a further ten amplifiers which are otherwise patched as summers. Ten more amplifiers are used as summers, and the remaining ten may be patched in special functions or as inverters. Up to three servo multipliers or resolvers may be plugged in each rack. The racks each contain capacity for thirty Digital Coefficient Units (DCUs), which are output devices on the PDP-11/20's DIGIBUS. Normal coefficient potentiometers are also available in each of the racks if required. Other DIGIBUS output devices are sixteen logic signals in each rack, and the S-bus and IC/CAL-bus, the S-bus consisting of sixteen lines bussed from sixteen Digital to Analogue Converters (DACs) to relay switchable outputs in each rack, and the IC/CAL-bus comprising a single bus, with outputs in each rack, originating at a single DAC. The DACs are located in rack 10, the I/O rack.

The MAC has an access system, which can access any amplifier output, differential check point, power supply rail or thirty special (rear-patchable) inputs in any rack, under control of the DIGIBUS. The PDP-11/20 controls an Analogue to Digital Converter (ADC) and 512-channel input multiplexer (MUX), with a conversion rate of 30,000 conversions per second. Other capabilities include the provision (through the DIGIBUS) of overload sensing for each amplifier, and for detection of amplifier oscillation or servo trip-out errors in each rack. The DIGIBUS provides control of the State of each of the racks, plus Mode control for each rack and for each integrator.

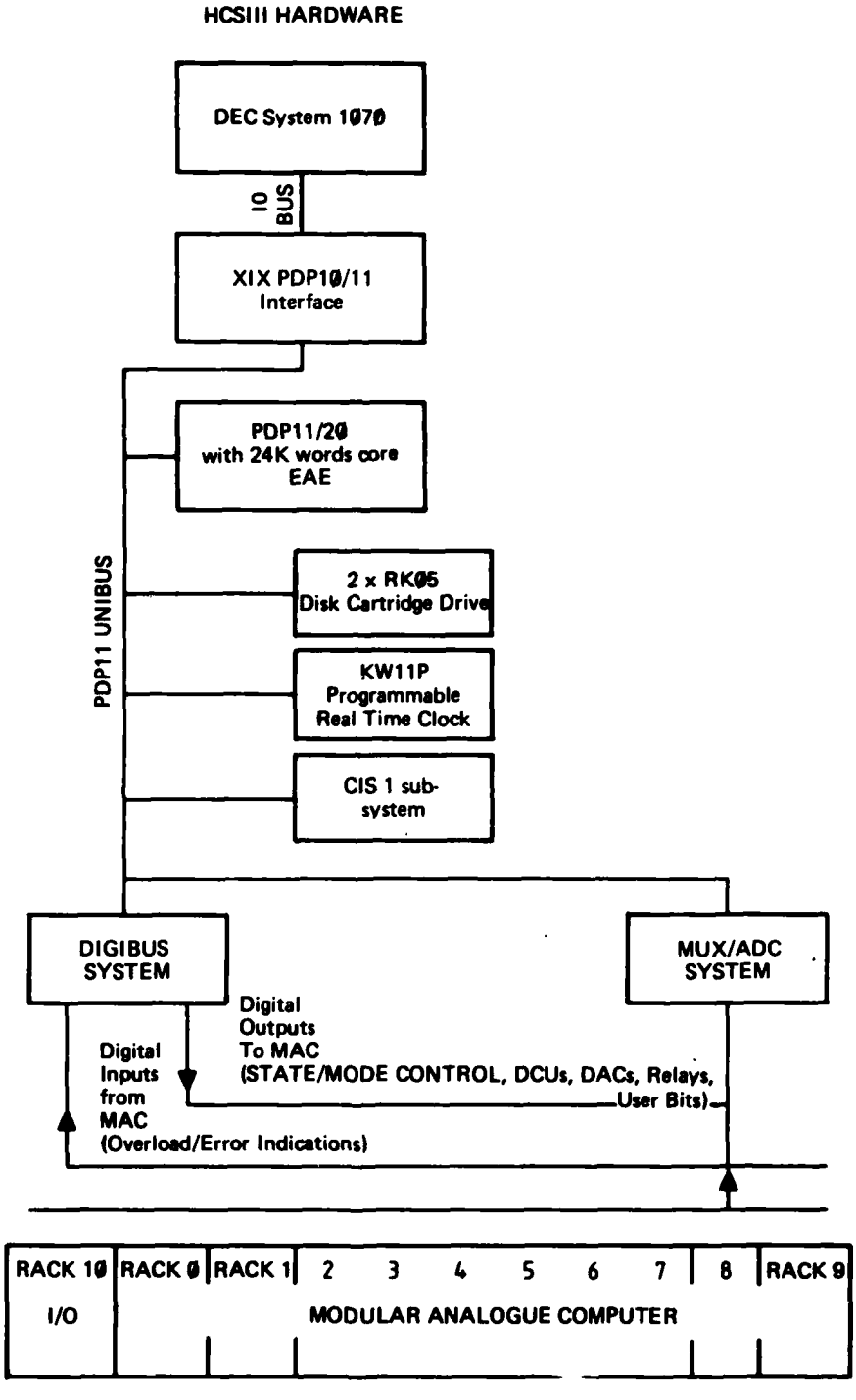


FIG. 1. HCS3 HARDWARE CONFIGURATION

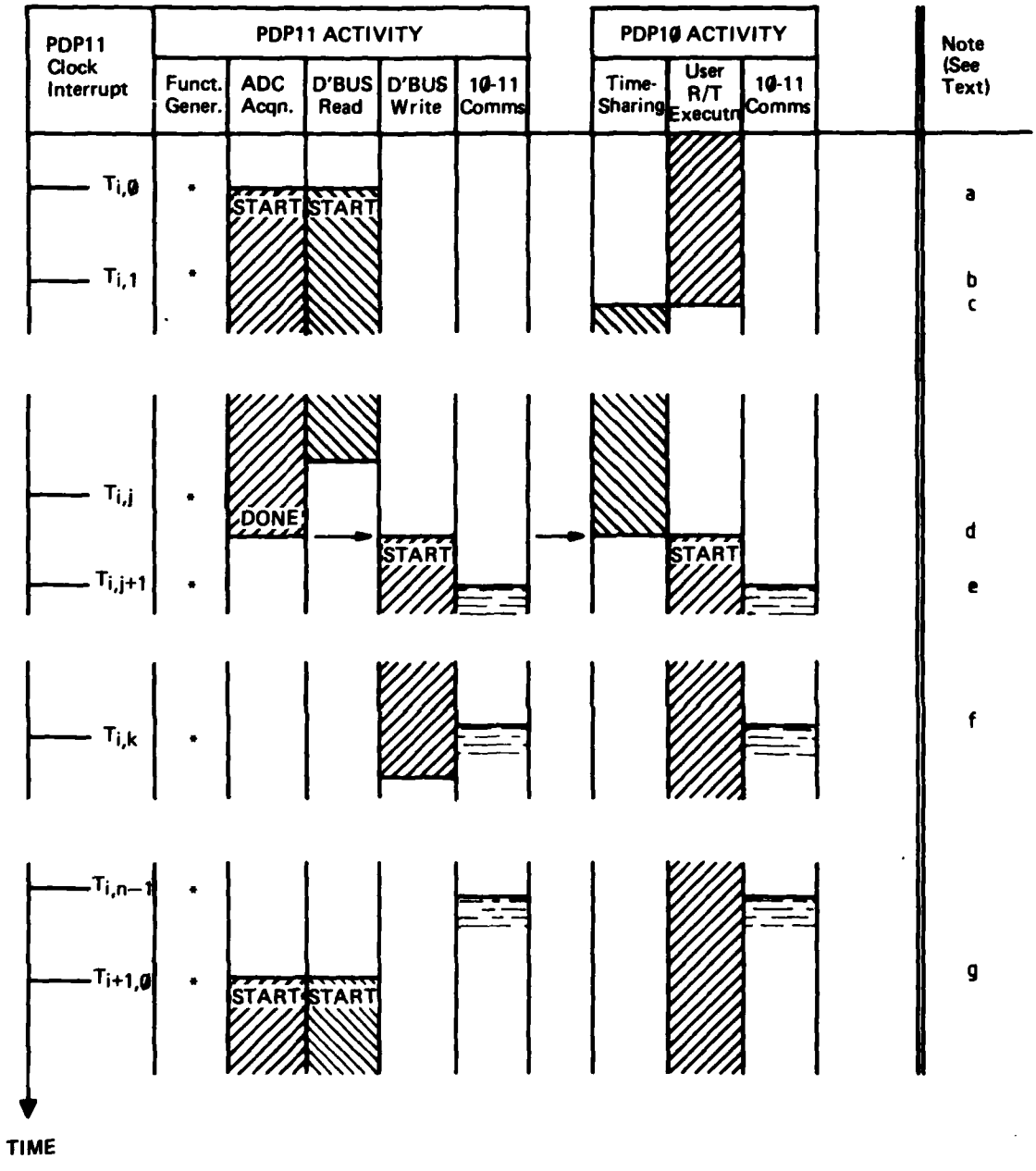


FIG. 2. ITERATION TIMING

2. OPERATION

2.1 Iteration Timing

The digital part of hybrid computation is performed, in the DECsystem-10 program, in two parts—a timesharing (T/S) part, and a real time (R/T) part. The R/T part is executed as interrupt level code in response to regular interrupts generated, at a user specified rate, by the (PDP-11/20's) programmable real time clock. Generation of analogue functions of single independent analogue variables is performed by the PDP-11/20, semi-autonomously of the DECsystem-10, at or at an harmonic of the iteration rate of the R/T computations in the DECsystem-10.

Iteration timing is illustrated in Figure 2, which shows the salient features of a complete iteration cycle, consisting of a number, n, of periods between interrupts of the PDP-11/20 real time clock. Bracketed references are indicated in the Notes column in Figure 2.

At (*a*), the PDP-11/20 clock interrupt determining the start of the ith iteration occurs. This, and subsequent clock interrupts, causes an update of the function generators to be performed. The PDP-11/20 software then commences acquisition, through its Analogue to Digital Converter (ADC), of all the analogue variables of interest in the problem, and commences a scan of the MAC Digital Serial Bus (DIGIBUS) input devices in the MAC. The Figure shows at (*b*) that these input activities may be interrupted for a function generator update. Execution of user R/T code in the DECsystem-10 may still be proceeding, though it must end, with normal timesharing jobs resuming execution, (*c*), before the next cycle's computations are due to start. This happens at (*d*), when both ADC and DIGIBUS read operations are completed. At this time, the PDP-11/20 commences those DIGIBUS write operations invoked by data transfers from the DECsystem-10 in the last cycle, and causes a DECsystem-10 interrupt which results in the commencement of this cycle's R/T code execution.

Communication of data between the DECsystem-10 and PDP-11/20 takes place, for example at (*e*), as burst mode transmissions, and at (*f*) one of these has caused a function generator update to be held over for the duration of the transmission.

At (*g*) the first real time clock interrupt of the next, $i+1$ th, cycle is shown. Once again the PDP-11/20 starts reading ADC and DIGIBUS data, and the cycle starts again. . . .

2.2 Program T/S Part Operation

With H3PAC, the processes of initializing hybrid operation and controlling cyclic execution of the user's R/T code are separated. Hybrid operation is initialized with (T/S level) subroutine HYBINI (see below for a full description of all the H3PAC subprograms) and terminated, with reversion to normal timesharing status, with (T/S level) subroutine HYBOFF. HYBINI puts the job into hybrid ready state. The program then has control of the MAC, and may perform pre-run conditioning, such as setting Digital Coefficient Units (DCUs) and Integrator Initial Conditions (IC). HYBINI automatically clears the PDP-11/20 digital function generators data tables, which may then be loaded. In this hybrid ready state, there is no cyclic execution of the user's R/T code.

The job may progress from the hybrid ready state into the hybrid iterate state using the (T/S level) subroutine ITERAT. In hybrid iterate state the user's R/T code, whose location is specified in the ITERAT call, is repetitively executed, as an interrupt routine, at a rate also specified in the ITERAT call.

Execution of the job's T/S part continues after a call to ITERAT, subject to interruptions arising from DECsystem-10 Priority Interrupt (PI) System activity (including the execution of the job's own R/T part) and to normal timesharing job scheduling by the Monitor. The T/S part may, by calling subroutine STOPIT, cause the job to revert to hybrid ready state, and a new ITERAT call, specifying a different iteration rate and piece of user R/T code if desired, may be made. Alternatively, it may use the (T/S level) subroutine PAUSIT to cause a temporary suspension of the cyclic execution of the user R/T code, leaving the job paused in hybrid iterate state. R/T execution can be continued using (T/S level) subroutine CONTIT.

The T/S part also has a subroutine TSLEEP which can be used to suspend its own activity until reactivated, either by its own R/T part, or by command from the controlling terminal.

2.3 R/T Part Operation

The R/T part performs computations using data obtained from the PDP-11/20, and may return results to the PDP-11/20 for transmission to the MAC. Since it is executed as an interrupt routine it may not perform normal I/O—this must be done with the T/S part of the program. The R/T part therefore has a subroutine, WAKETS, which can be used to reactivate the T/S part from a TSLEEP call, and pass it an argument, to enable this process to be synchronized.

The start of the R/T part is identified in the ITERAT call using the DECsystem-10 FORTRAN IV convention for passing statement numbers as arguments of subprograms. There may be several parts of the program used as the R/T part at separate times during the execution of the program, only one can be the R/T part at any given time. The execution of a R/T part cycle ends with a call to subroutine RETNIT, which dismisses the interrupt responsible for the R/T cycle's execution. An argument of RETNIT may be used with the same effect as the T/S level subroutine PAUSIT, to cause temporary suspension of execution of subsequent R/T part cycles.

3. MAC DATA AND CONTROL

3.1 General

HCS3 manages the data transfers between the analogue and digital parts of the hybrid system, and provides the capability for logging the data acquired from the analogue part on the PDP-11/20's RK05 disks. The PDP-11/20 controls all the hybrid interface devices.

3.2 Hybrid Computation Data

Data originating in the MAC are made up of analogue variables, which are converted to digital form by the ADC, and overload/error signals obtained through DIGIBUS input operations. H3PAC provides subprograms allowing access to any of these data to be obtained either by the T/S part of the program in hybrid ready state or by the R/T part. It is not available to the T/S part in hybrid iterate state, and in any case would not be useful because of the lack of control of timing there.

The MAC's access system output, at several scalings and processed by a peak-to-peak noise detecting network, is hardwired to ADC input channels. The access values are also available to the DECsystem-10 program, as described in the preceding paragraph. This process is described in Section 9.2 below.

The DECsystem-10 program can pass variables and logic signals to the MAC via the PDP-11/20 and the DIGIBUS, using H3PAC subprograms. Variables are output through the DCUs, the MAC has provision for 30 of these per rack, and logic signals are output as MAC user logic bits, 16 per rack.

Again, this may be done from the T/S part in hybrid ready state, or from the R/T part, and may not be done from the T/S part in hybrid iterate state.

3.3 Data Logging

The PDP-11/20's RK05 removable cartridge disk drives can be used for logging MAC variables during the course of a hybrid computation. Up to 475 user nominated plus the four access system output ADC channels, plus the 32 DIGIBUS input words plus the ADC over-range flag can be logged (a total of 512 quantities), provided that real time constraints are not exceeded. Data is logged every R/T cycle. The data may subsequently be read back into the DECsystem-10 for further processing.

Data are logged under control of the DECsystem-10 program. Logging may be turned on and off by calls to H3PAC subprograms made from the T/S part in hybrid ready state or from the R/T part. If turned on in hybrid ready, logging does not actually commence until iterative execution of the R/T part is started.

3.4 MAC Control

H3PAC subprograms give the DECsystem-10 program control of MAC state and mode settings, the S-bus relays in each MAC rack, and the S-bus and IC/CAL-bus DACs. These can be used in the R/T part, or in the T/S part in hybrid ready state.

A subroutine callable only from the T/S part in hybrid ready state provides for setting of initial conditions to integrators patched to the IC/CAL-bus. Each integrator is left in HOLD mode, with its rack in COMPUTE state, after it has been set up to the specified initial condition.

4. H3PAC SUBPROGRAMS

Table 4.1 contains a list of the subprograms provided by H3PAC, indicating whether they may be used in the R/T or T/S parts, and, if the latter, whether they may be used in hybrid ready or hybrid iterate states.

The subprograms' functions and parameter lists are described in succeeding Chapters.

TABLE 4.1

H3PAC Subprograms

Name	Type*	T/S Ready	T/S Iterate	R/T Part	Function
HYBINI	S				Enter Hybrid ready
HYBOFF	S	Y	Y		Revert to normal T/S
ITERAT	S	Y			Enter Hybrid Iterate
STOPIT	S		Y		Leave Hybrid Iterate
PAUSIT	S		Y		Suspend R/T cycles
CONTIT	S		Y		Continue R/T cycles
TSLEEP	S		Y		Wait to be woken
WAKETS	S			Y	Wake T/S part
RETNIT	S			Y	End of this iteration
INTINI	S	Y			Load Integrator I/Cs
FGDATA	S	Y			Load Digital F/G data
DCUVAL	S	Y		Y	Send data to DCUs
USRBIT	S	Y		Y	Set/Clear user bits
ADCRED	S	Y		Y	Read ADC Channels
STATE	S	Y		Y	Specify MAC state
MODE	S	Y		Y	Specify MAC mode
TIMSCL	S	Y		Y	Specify MAC time scaling
ACCADR	S	Y		Y	Send access address
ACCVAL	S	Y		Y	Read access outputs
MACERR	F	Y		Y	Read DIGIBUS error flag
ADCERR	F	Y		Y	Read ADC overrange flag
ERREAD	S	Y		Y	Read DIGIBUS error words
LOGGO	S	Y		Y	Start data logging
LOGSTP	S	Y		Y	Stop logging
LOGSER	F	Y		Y	Get logging serial
FGCONN	S	Y		Y	Connect digital F/G
SRELAY	S	Y		Y	Pick/Drop S-bus relays
SVALUE	S	Y		Y	Send S-Bus DAC values
ICBVAL	S	Y		Y	Send data to IC-Bus DAC
CPFRAC	S				Set CPU use limit
HSTATS	S	Y	Y		Read R/T timings

* S = subroutine: F = function.

5. PROGRAM OPERATION CONTROL SUBPROGRAMS

This group of subprograms has members which may be called only from the T/S part and others which may be called only from the R/T part. They provide the programmer with the means to control the progression of the program into the various stages of hybrid operation. The subprograms are described below.

5.1 Subroutine HYBINI (addr array, addr count, data array, scale array, data count, rack array, log, ident)

5.1.1 Purpose

HYBINI puts a normal timesharing job into hybrid ready state. The job enters a high priority run queue (HPQ), locks itself in core, and gains control, via the DECsystem-10 real time trap (RTTRP) features, of the XIX interface. This enables it to control the PDP-11/20, and through that the MAC, which it can then use for hybrid computation.

HYBINI causes a RESET command to be sent out on the DIGIBUS, and this initializes the MAC. It clears the digital function generator tables in the PDP-11/20's core. A HYBINI argument is used to stipulate whether data logging on the PDP-11/20's disks is to be done; if it is then logging is initialized, and HYBINI checks that both disk drives have a disk loaded and write-enabled then writes a preamble record on drive zero at disk address zero (see Chapter 11, below). This record contains ASCII identification, supplied as the last of the HYBINI arguments, plus a description of the variables that will be logged, when logging is enabled, consisting of a count and a list of the ADC channel numbers of the variables.

HYBINI's other arguments are used to specify the 'active' analogue quantities in the problem. These, plus the access system inputs and the DIGIBUS inputs, are logged on the PDP-11/20's disks if and when logging is enabled. A subset, or all, of the active variables may be read by the DECsystem-10 program during hybrid program execution. Further HYBINI arguments specify which ones may be thus required, and supply scale factors for them. This means that if the value of an analogue variable may at some stage of the computation be required in the digital part, then that variable must be made active, and be made a member of the set which may be read by the DECsystem-10, in the HYBINI call. If the quantity will never be required by the digital part of the computation, but is of interest for logging, then it need only be made active. The output of the access system, at its various gains, need not be made active with HYBINI —these variables are always available, and will always be logged when logging is enabled.

5.1.2 Arguments

HYBINI has eight arguments, which are:

(1) *addr array*

Type: Single Precision* Array.

Use: To specify the MAC address at which the variables to be made active in subsequent hybrid operation are computed.

Format: Active variables are specified in successive entries in the array by their MAC amplifier address as an ASCII literal having the form 'Armnc', where A indicates that it is an amplifier address, where r is the amplifier rack number, in the range 0 to 9, where mn is the amplifier number in the rack, in the range 00 to 39, and where c is a code determining the converter scale at which the variable is to be read, and the level of hardware overrange detection to be enabled at that time. The values c may have, and their meanings, are:

(i) c = blank (i.e. omitted) is the default setting, and indicates that the quantity is to be read with an ADC scale of 100V (nominal), with hardware ADC overrange detection enabled;

* i.e. REAL, INTEGER or LOGICAL.

(ii) c = S (Scaled) stipulates that the variable is to be read using the ADC 10V (nominal) scale, with hardware overrange detection enabled;

(iii) c = X (no mnemonic) indicates that the variable is to be read at an ADC scale of 10V (nominal), and with hardware overrange detection disabled.

This addressing scheme covers only those variables computed by MAC amplifiers. To make a variable patched to one of the ADC system patchable inputs active, the address is specified in the form 'Ilmnc'. The leading I indicates a patchable Input, lmn is the input channel number in ASCII in the range 000 to 511 (with leading zeroes optionally replaced by blanks), and c is a code having the same range of values and meanings as just described.

(2) *addr count*

Type: INTEGER Scalar.

Use: This argument is used to specify the number of variables being made active by the call, that is, the number of entries in *addr array*.

Format: An INTEGER in the range 0 to 475 (decimal). This allows for the 32 DIGIBUS inputs, the ADC overrange word, and the four access system values to be accommodated in the maximum of 512 words loggable each cycle.

(3) *data array*

Type: REAL Array.

Use: This argument, plus the other two yet to be described, is used in specifying to the software the details of those active analogue variables that may be required for computation in the digital part. This array provides locations into which the software will place data in response to calls to subroutine ADCRED (see Section 8.3 below). Elements of this array are in one-to-one correspondence with elements of *addr array*, up to the limit of this array. Therefore for efficient use of memory the subset of the active analogue variables that may be required to be read by the digital part should be specified first in *addr array*, so that this *data array* need then only to be large enough to accommodate this subset. Where feasible, variables likely to be required at the same time should be grouped together, to minimize the number of calls to ADCRED required to obtain their values. The software ignores and does not change data stored in this array at the time of a call to HYBINI.

Format: REAL numbers representing variable values in physical units.

(4) *scale array*

Type: REAL Array.

Use: Elements of this array, in one-to-one correspondence with the elements of *data array*, and hence with the leading elements of *addr array*, are loaded with the scale factors of the analogue variables to which they thus correspond. When the program uses ADCRED to obtain variable values, the incoming ADC readings, in converter units (a number in the range $-1{\cdot}0$ to $+1{\cdot}0$ nominal) are multiplied by the corresponding scale factors before being stored in the appropriate locations in *data array*. The software does not attempt to compensate for ADC scale setting.

Scale factors in *scale array* may be changed at any time. It is unwise to change a scale factor in the T/S part in hybrid iterate state unless timing is synchronised with the R/T part.

Scale factors are not supplied to the PDP-11/20, and quantities logged on its disks are direct ADC binary outputs.

Format: Entries are REAL numbers.

(5) *data count*

Type: INTEGER Scalar.

Use: This argument contains the number of active variables for which *data array* and *scale array* contain entires.

Format: An integer number in the range zero to the number contained in *addr count*.

(6) *rack array*

Type: LOGICAL Vector of dimension 10 (decimal).

Use: Each element of *rack array* is set by the software to .TRUE. if its corresponding MAC rack is powered up, and to .FALSE. if it is not. For Hybrid operation, Rack 10 of the MAC must be powered up, so entries are provided only for the ten computing racks, 0 to 9.

(7) *log*

Type: LOGICAL Scalar.

Use: If *log* has the value .TRUE., then the user wishes to have the option of using the PDP-11/20's disks for data logging later in the program, and logging will be initialized and a preamble record written.

(8) *ident*

Type: ASCII string of 100 characters (20 words).

Use: This argument is used to supply identification for the data to be logged on the PDP-11/20's disks. If *log* is .TRUE., then this string is copied into the preamble record on the disks.

5.1.3. Errors

HYBINI detects both operation and data errors. Operation errors may be fatal, as in the case of a user without RTTRP privileges attempting a HYBINI call, or they may be recoverable, say PDP-11/20 power down. Data errors, such as meaningless ADC address specifications, are invariably fatal. In each case an appropriate error message is typed on the job's controlling terminal, and for recoverable errors the option of attempting to proceed after taking remedial action is proffered to the user.

5.1.4 Fixed ADC Channel Allocations

The MAC amplifier outputs occupy ADC system input channels 0 to 399 (inclusive), channels 400 to 489 are available for patchable inputs, and the remaining channels are allocated as follows:

Channels 490–505 to the S-bus DACs 0–15 Outputs;
Channel 506 to the IC/CAL-bus DAC Output;
Channels 508 and 509 to the − and + Reference Supply;
Channels 507, 510 and 511 to the access system output, respectively directly, through 100 gain, and through the noise detector.

This information is required if any of these quantities are needed in a computation and must be made active variables in the program.

5.2 Subroutine HYBOFF

5.2.1 Purpose

HYBOFF returns the job from hybrid ready or iterate state to normal timesharing status. The XIX interface is released and the job unlocks itself and enters normal run priority condition. A HYBOFF call causes a RESET command to be sent out on the DIGIBUS.

HYBOFF has no arguments.

5.2.2 Errors

HYBOFF detects the fatal error of calling it from the R/T part of the program.

5.3 Subroutine ITERAT (period, subperiod, *rtcode)

5.3.1 Purpose

This subroutine may only be called from the T/S part of the program in hybrid ready state. It causes the repetitive interrupt level execution of the user's R/T code to commence, and the job to enter hybrid iterate state. Its arguments specify the iteration period, the PDP-11/20 function generator update period, and the location of the start of the user's R/T code.

More than one call to ITERAT may be made in a given program, and hence more than one section of R/T code may be used (at different times), provided that the job is returned to hybrid ready state with subroutine STOPIT (see Section 5.4 below) between the calls to ITERAT.

5.3.2 Arguments

Three arguments must accompany a call to ITERAT:

(1) *period*

Type: REAL Scalar.

Use: This argument specifies the hybrid computation digital program R/T part repetitive execution period.

Format: The period is specified as a REAL number of seconds.

Limits: Period must lie in the range 0·005 to 2·5 seconds.

(2) *subperiod*

Type: REAL Scalar.

Use: The PDP-11/20 digital function generator update period is specified by *subperiod*. Its value must be an integral divisor of the previous argument, the problem iteration *period*.

Format: A REAL quantity in seconds.

Limits: The subperiod must lie in the range 0·005 to 0·65 seconds, specified to a resolution of 10 microseconds.

(3) **rtcode*

Type: FORTRAN STATEMENT NUMBER (prefixed with * or $).

Use: This argument is used to specify to the software the location of the start of the user's R/T code to be executed during subsequent hybrid iterations.

Format: A FORTRAN STATEMENT NUMBER, using the DECsystem-10 FORTRAN 10 (F10) Compiler convention that the number be prefixed by * or $.

5.3.3 Errors

As well as errors in range of its first two arguments, ITERAT detects if it has been called from the T/S part in hybrid iterate state or from the R/T part. These errors are all fatal to program execution.

5.4 Subroutine STOPIT

5.4.1 Purpose

STOPIT, called from the T/S part in hybrid iterate state, causes cessation of the repetitive execution of the user's R/T code, and hence causes the job to return to hybrid ready state.

5.4.2 Arguments

Subroutine STOPIT uses no arguments.

5.4.3 Errors

STOPIT detects only fatal errors resulting if it is called from the R/T part or from the T/S part in hybrid ready state.

5.5 Subroutine PAUSIT

5.5.1 Purpose

Subroutine PAUSIT may be called from the T/S part of the program, in hybrid iterate state, to produce a temporary suspension of the repetitive execution of the R/T part. The pause takes place after the next time that the ADC has completed its inputs and the DIGIBUS outputs have been done. Referring to Figure 1, if a PAUSIT were called from the T/S part in the time between (*c*) and (*d*), then the user R/T code execution shown commencing at (*d*) would not take place until a subsequent call was made to CONTIT (see below), although the DIGIBUS output indicated starting there would be done.

PAUSIT synchronizes itself with the actual time at which the pause becomes effective, and does not return until then.

The job remains in hybrid iterate state, but logging on the PDP-11/20's disks stops while the job is paused.

PAUSIT has no effect on the operation of function generators in the PDP-11/20.

5.5.2 Arguments

Subroutine PAUSIT has no arguments.

5.5.3 Errors

PAUSIT detects the fatal errors resulting from calling it from the T/S part when not in hybrid iterate state, or from the R/T part.

PAUSIT has no effect if the R/T part is already paused, either because of a previous call to PAUSIT or because of the use of the RETNIT argument (see below).

5.6 Subroutine CONTIT

5.6.1 Purpose

This subroutine, called from the T/S part of the program in hybrid iterate state, causes the cyclic execution of the program's R/T part to resume from the pause induced by a prior call to PAUSIT or by use of the RETNIT argument (see below).

If the R/T part is not paused then CONTIT has no effect.

5.6.2 Arguments

CONTIT has no arguments.

5.6.3 Errors

Fatal errors detected by CONTIT result from calling it from the T/S part in hybrid ready state or from the R/T part.

5.7 Subroutine TSLEEP (reason)

5.7.1 Purpose

TSLEEP provides a means by which the T/S part of the program can become dormant, awaiting either a signal to proceed from the R/T part or activity, specifically the typing of a break character (RETURN or ESCAPE), at the controlling terminal.

Typically the T/S part of the hybrid job has little to do once the R/T part has started its iterative execution, except perhaps for performing normal I/O. TSLEEP lets the T/S part withdraw from contention with other jobs for run time.

Its argument allows non-ambiguous and synchronized communication with the R/T part.

Execution of the T/S part resumes when TSLEEP returns, having received one or other of the signals mentioned above.

5.7.2 Argument

TSLEEP returns one argument, which is described below:

(1) *reason*

Type: INTEGER Scalar.

Use: *reason* is used for non-ambiguous communication from the R/T part to the T/S part. It returns, in the T/S part, a value which can be used by the program to interpret the reason for the job's having returned from the TSLEEP call (namely Terminal activity or a signal from the R/T part). A value of −1, FORTRAN logical .TRUE., is returned when user teletype activity has caused TSLEEP to return. When TSLEEP returns as a result of a R/T part call to subroutine WAKETS with a positive integer argument, *reason* returns that positive integer.

It is disconnected, by the software, from the WAKETS argument, so that the R/T part may change that value, after a WAKETS call, without affecting the value of *reason* returned by a TSLEEP call.

It is poor programming practice to use the same FORTRAN variable as the argument to both TSLEEP and WAKETS. The results are predictable, but doing it suggests a poor grasp of hybrid programming, and in turn that the results will not be what is intended.

5.7.3 Errors

The only errors detected by TSLEEP are the fatal errors of calling it from the T/S part in hybrid ready state or from the R/T part.

5.8 Subroutine WAKETS (reason)

5.8.1 Purpose

Subroutine WAKETS is called from the R/T part of the program, and causes the T/S part to return from a TSLEEP call. Its argument may be used to communicate with the T/S part, as described in the previous section.

If WAKETS is called from the R/T part while the T/S part is active, that is not dormant inside a call to TSLEEP, then a subsequent T/S part call to TSLEEP will return at once—the software remembers the WAKETS call. However, only one, the most recent of them, is remembered.

5.8.2 Arguments

WAKETS has one argument, described below:

(1) *reason*

Type: INTEGER Scalar.

Use: This argument supplies the value which will subsequently be returned to the T/S part when TSLEEP returns.

Format: *reason* must be greater than zero; zero and negative values are illegal.

5.8.3 Errors

A fatal error results if WAKETS is called from the T/S part of a program, or is called with a zero or negative argument.

5.9 Subroutine RETNIT (pause)

5.9.1 Purpose

Subroutine RETNIT is called in the R/T part to signal to the software that the user's R/T code execution has been completed this cycle. There is no return from a RETNIT call; instead, control passes to H3PAC, where accounting and various housekeeping tasks are performed and finally the interrupt dismissed. As indicated above, RETNIT's argument can be used to induce a pause in R/T cycling analogous to that resulting from a T/S level PAUSIT call.

5.9.2 Argument

RETNIT has one argument:

(1) *pause*

Type: LOGICAL Scalar.

Use: When RETNIT is called with *pause* having the value .TRUE. (or a negative numerical value), repetitive execution of the user's R/T code is suspended in the way described for PAUSIT above. It can be resumed with a CONTIT call from the T/S part. If *pause* is .FALSE. (numerically zero or positive) there is no effect.

5.9.3 Errors

RETNIT detects the fatal error resulting from calling it from the T/S part.

6. HYBRID SETTING-UP SUBPROGRAMS

The two subprograms in this category are called from the T/S part in hybrid ready state (that is after the HYBINI call). They are used in setting-up the MAC Integrator Initial Conditions values, and to load the PDP-11/20 Function Generator (F/G) data tables.

6.1 Subroutine INTINI (addr array, data array, scale array, count)

6.1.1 Purpose

INTINI is used for setting up MAC INTegrators INItial Conditions values. It starts with a list of Integrator addresses, a list of data, and a list of scale factors. The subroutine works its way through these lists, for each triplet of entries, it ensures that the rack containing the integrator addressed by the *address array* entry is in COMPUTE state, and puts the integrator into INITIAL CONDITION (IC) mode. It then calculates the appropriate value for transmission on the MAC IC/CAL-bus from the entries from *data array* and *scale array*. This value is then sent to the IC/CAL-bus DAC, and held there for one second to allow the integrator to settle. The integrator is then switched into HOLD mode, and its output read, using the MAC access system, to verify that the integrator has indeed taken up the correct IC value.

Note that each integrator to which an IC setting is to be transmitted with INTINI must have its IC input patched to the IC-bus.

In this process, rack 10, which supplies REFERENCE levels to the other racks, will be put and left in COMPUTE state.

6.1.2 Arguments

INTINI uses four arguments, described below:

(1) *addr array*

Type: Single Precision Array.

Use: Used to specify the MAC integrators whose Initial Conditions are to be set up by this call.

Format: Integrators are identified by their ASCII addresses, in the form 'Armn', where the 'A' is the usual way in this software of indicating an amplifier, where 'r' is the rack number (in the range 0 to 9), and where 'mn' is the integrator number (in the range 10 to 19). This type of entry can readily be generated as a FORTRAN ASCII literal.

(2) *data array*

Type: REAL array.

Use: The quantities in this array, in one-to-one correspondence with the *addr array* entries, supply the Initial Conditions values to be set up.

Format: *data array* entries must be expressed as REAL numbers, in physical units, equal to the required initial value of the variable. The software scales the value to machine units, and looks after the IC sign inversion.

(3) *scale array*

Type: REAL Array.

Use: These entries specify the analogue problem scalings of the variables computed by the integrators addressed in *addr array*. The software uses these scalings to compute the machine unit IC settings.

Format: REAL numbers.

(4) *count*

Type: INTEGER Scalar.

Use: *count* specifies the number of integrators to have ICs set as a result of this call.

Format: INTEGER number.

6.1.3 Errors

All errors detected by INTINI are fatal. They may result from calling it from the T/S part in hybrid iterate state, or from the R/T part, or from errors in the address formats or attempts to exceed the integrator IC range of $-1{\cdot}0$ to $+1{\cdot}0$ machine units, or from failure of the hardware to respond to the setting either as a result of patching errors or malfunctions. In each case, an appropriate error message is given on the controlling terminal.

6.2 Subroutine FGDATA (table no, data array)

6.2.1 Purpose

If the PDP-11/20 function generator capability is to be used, the data tables must be loaded, and this subroutine is provided to perform that function. Each call to FGDATA loads one table, identified by a number which is later used in specifying the connection of function generators.

FGDATA may only be called from the T/S part of the program in hybrid ready state, that is after the HYBINI call, as HYBINI clears the function generator tables. It may not be used in the T/S part during hybrid iterations or in the R/T part.

There is no need to load unused data tables, should a function generator use a table which has not been loaded, the DCU specified will always output a zero coefficient.

6.2.2 Arguments

FGDATA requires two arguments:

(1) *table no*

Type: INTEGER Scalar.

Use: This argument conveys to the software the identification number of the function generator table being loaded by this call. Twenty tables may be loaded, numbered from 0 to 19.

(2) *data array*

Type: REAL Vector of 129 entries.

Use: *data array* carries the 129 entries for the function generator data table being loaded by this call. Entries have the form of REAL numbers, in the range 0·0 to 16·0, specifying actual DCU coefficient settings at the endpoints of the 128 intervals defining the function.

6.2.3 Errors

Fatal errors result if this subroutine is called from the R/T part, or from the T/S part in hybrid iterate state, or if any of the parameters are out of range.

7. HYBRID DATA MANIPULATION SUBPROGRAMS

These subprograms provide the means by which data are transferred between the digital and analogue parts of the hybrid computation. They may be called from the T/S part of the program in hybrid ready state, or from the R/T part. Calling them from the T/S part in hybrid iterate state will cause a fatal error.

When called from the T/S part, these subprograms execute to completion, by causing the execution of R/T cycles in which no user code is actually executed. This means that on return from a T/S part (in hybrid ready state) call to subroutine DCUVAL (see below for a description) which sends values to DCUs, the settings will have actually been output to the DCUs. Similarly, when reading ADC input data from the MAC, the software will invoke R/T cycles (hidden from the user) to ensure that the values returned are current.

When called from the R/T part, the timing of the execution of the data transfers arising from these subprograms is as was described in Section 2.1. In a single iteration's execution of the R/T part, a maximum of 512 DIGIBUS output data transfers may be performed. This limit is determined by the size of a core buffer in the PDP-11/20. If the limit is exceeded, the software treats it as a fatal error, stopping the job and typing a warning message. This limit is not particularly severe for most applications.

7.1 Subroutine DCUVAL (addr array, data array, scale array, count)

7.1.1 Purpose

DCUVAL is used for sending coefficient values to DCUs. DCU settings made this way remain fixed until changed by a subsequent call to DCUVAL, or until all DCUs are cleared to zero by the DIGIBUS RESET generated in response to a call to HYBOFF at the end of hybrid operations.

DCUs used by PDP-11/20 function generators may be loaded with an initial value using DCUVAL but, as described in Chapter 12, below, the function generators will cycle, albeit at a non-synchronous rate, before hybrid iterate state is entered so that start-up transients can be minimized.

The current settings of all the DCUs are stored inside the DECsystem-10 software to enable redundant data transfers to be suppressed as far as possible.

7.1.2 Arguments

DCUVAL has four arguments:

(1) *addr array*

Type: INTEGER Array.

Use: Used to specify the DCUs to be loaded as a result of this call.

Format: DCU numbers are specified as decimal integers of the form rmn, where r is the rack number (0 to 9), and mn is the unit number in the rack (0 to 29).

(2) *data array*

Type: REAL Array.

Use: To specify settings for the DCUs indicated in *addr array*.

Format: REAL physical values of the coefficients required (entries are in one-to-one correspondence with the entries in *addr array*).

(3) *scale array*

Type: REAL Array.

Use: To supply coefficient scaling information, to enable conversion between the physical coefficient entries in *data array* and the machine coefficients actually sent to the DCUs.

Format: REAL coefficient scale factors. To explain this best, consider the use of a DCU in the conversion of an angular quantity from radians (in which it is assumed generated) to degrees (say for driving a piece of system hardware). The physical equation is

$$D = 180/\pi * R,$$

so the physical coefficient value, which would be an entry in *data array*, is $180/\pi$, or $57 \cdot 2 \ldots$

Suppose further that the scaling for D is d, and for R is r. The machine equation becomes

$$d\hat{D} = 180/\pi * r\hat{R}, \text{ or,}$$

$$\hat{D} = (r/d) * 180/\pi * \hat{R}.$$

In this case the entry in *scale array* would have the value r/d. It is essential that the actual number sent to the DCU is a positive number, less than or equal to $16 \cdot 0$. That is, the quantities entered in *scale array*, when multiplied by their corresponding *data array* entries should produce products in the range $0 \cdot 0$ to $16 \cdot 0$.

(4) *count*

Type: INTEGER Scalar.

Use: This variable contains the INTEGER number of DCUs being loaded by this call.

7.1.3 Errors

All errors detected by DCUVAL are fatal, causing the program execution to terminate with the printing of the appropriate error message at the job's controlling terminal. The software detects if DCUVAL is called from the T/S part in hybrid iterate state, checks the validity of the DCU addresses in *addr array*, and checks the data by ensuring that the physical coefficients derived by multiplying the data and scale factor entries are in the range $0 \cdot 0$ to $16 \cdot 0$.

7.2 Subroutine USRBIT (value, addr array, count)

7.2.1 Purpose

USRBIT provides the means for setting or clearing any of the DIGIBUS user logic outputs in the MAC. There are sixteen separate bits in each rack. They change only in response to calls to this subroutine, and are cleared by the DIGIBUS RESETs generated at hybrid initialization (HYBINI) and termination (HYBOFF).

To save XIX interface transactions and PDP-11/20 processor time, the settings of the user bits are stored in a buffer in the DECsystem-10 software. Changes caused by a USRBIT call are pooled where possible, so that a call which sets or clears all sixteen bits in a MAC rack uses only one XIX data transfer. The setting and subsequent clearing of a single user bit in successive calls to USRBIT is not suppressed, and will cause a pulsed output. Redundant transfers are suppressed, so that a request to set an already set bit, or to clear a cleared one will be ignored.

7.2.2 Arguments

There are three arguments for USRBIT, as follows:

(1) *value*

Type: LOGICAL Scalar.

Use: This argument determines whether bits are to be set or cleared by this call. To set bits, *value* should be made .TRUE., and to clear them it should have the value .FALSE..

(2) *addr array*

Type: INTEGER Array.

Use: The addresses of the user bits to be set or cleared by this call are stored in this array.

Format: User bit addresses are stored as decimal INTEGER quantities of the form rsbc, where rs is the rack number and bc is the bit number. Both rs and bc must lie in the range 00 to 15.

(3) *count*

Type: INTEGER Scalar.

Use: *count* specifies the number of user bits, addressed by entries in *addr array*, that are to be set or cleared by this call.

7.2.3 Errors

USRBIT detects two types of fatal error—attempting to call it from the T/S part when not in hybrid ready state, and specifying an illegal user bit address in *addr array*.

7.3 Subroutine ADCRED (data array, count)

7.3.1 Purpose

The digital program uses ADCRED to input the values of the analogue variables that it needs for its computations. These variables must have been made active, and made members of the subset available to the digital part, in the HYBINI call prior to using ADCRED.

ADCRED may be called from the T/S part in hybrid ready state, or from the R/T part. In the former case, the software will stimulate an internal R/T cycle causing a new set of ADC data conversions to be made and then transfer the results requested back to the calling program. In the latter case (call from R/T part), the timing is such that the latest values of the variables read by the ADC will be passed to the DECsystem-10 and through to the calling program.

The values will be scaled by the scale factors attached to the *data array* by the *scale array* argument of the HYBINI call.

7.3.2 Arguments

ADCRED has two arguments, as follows:

(1) *data array*

Type: REAL Array or Array Element.

Use: This argument is used to specify the first of the group of analogue variables to be input by the call. It must be an element of the *data array* argument specified in the HYBINI call, or, if the group to be input starts with the first member of that array, it can be identical with the HYBINI *data array* argument.

ADCRED returns the (scaled) analogue quantities in successive elements of *data array*.

(2) *count*

Type: INTEGER Scalar.

Use: The calling program loads *count* with the number of analogue variables to be input by this call. There is a logical maximum limit of 475 for *count*, as this is the maximum number of variables that may be made active in the original HYBINI call.

7.3.3 Example

The use of ADCRED can be appreciated with the aid of a simple program example. Consider the following extracts from a program's code:

```
      REAL ACTADR(20), DATAIN(10), DATASC(10)
      DIMENSION DUMMY(10)
      DATA ACTADR/'Arnn', 'Arnn',..../
      DATA DATASC/ss.ss,tt.ttt,...../
      EQUIVALENCE (A,DATAIN(1)), (B,DATAIN(2))
      EQUIVALENCE (E,DATAIN(5))
      .
      .
      .
      .
1     CALL HYBINI (ACTADR,20,DATAIN,DATASC,10,
          DUMMY,.TRUE.,'100-character run ident')
      .
      .
      .
2     CALL ADCRED (DATAIN, 2)
      .
      .
      .
3     CALL ADCRED (E, 6)
      .
      .
      .
```

The declarations, plus the HYBINI call at FORTRAN statement number 1, indicate that the programmer wishes to make 20 analogue variables active: their addresses being given in the DATA statement for ACTADR. Of these 20, the first 10 are to be made available to the digital part of the program, with the scale factors given in the DATA statement for DATASC. Input from these ten variables will go into the elements of DATAIN, and the EQUIVALENCE statements indicate that the programmer wishes to refer to the first input as A, the second as B, and the fifth as E.

At statement number 2, the user has asked for the input of two variables, the DATAIN argument indicating that they are to be the first two. After this call the user will have new values in DATAIN(1) and DATAIN(2), which he may also refer to as A and B respectively. A paraphrase for this call would be:

```
2     CALL ADCRED (A, 2).
```

At statement number 3 the user has requested input from a second group of variables, in this case the six starting at E (equivalent to DATAIN(5)). On return from this call new values for E and DATAIN(6) to DATAIN(10) will have been obtained.

7.3.4 Errors

All errors detected by ADCRED are fatal: the program stops and an appropriate message is output on the job's controlling terminal. The software checks first that the call is legal, that is that it has been made from the R/T part or from the T/S part in hybrid ready state. The arguments are then checked to ensure that they call for inputs only into the *data array* specified in the HYBINI call. This is intended to minimize the risk of data corruption.

8. MAC CONTROL SUBPROGRAMS

These subprograms give the digital program control over the MAC's mode and state and integrator timescales. Like the Data Manipulation subprograms, they may be called from the T/S part of the program in hybrid ready state, or from the R/T part, and calling them from the T/S part in hybrid iterate state is detected as a fatal error.

Again, like the Data Manipulation subprograms, if they are called from the T/S part their execution is completed before they return; if they are called from a R/T cycle, their effect in the MAC hardware occurs after the completion of the next cycle's input conversions, in accordance with the timing described in Section 2.1.

The software keeps a continuous record of the mode and state settings in the MAC, to enable it to suppress redundant data transfers, and to detect erroneous requests.

8.1 Subroutine STATE (setting, addr array, count)

8.1.1 Purpose

Subroutine STATE is used for controlling the state settings of the MAC racks.

8.1.2 Arguments

STATE has three arguments:

(1) *setting*

Type: Single Precision Scalar.

Use: This variable is loaded by the caller with a code for the particular MAC state into which the racks specified by the call are to be put.

Format: The code takes the form of an ASCII string, of which only the leftmost character is interpreted as the initial letter of the desired state ('S', 'P', 'O', or 'C' for, respectively, STAND BY, POT SET, OP CAL or COMPUTE).

(2) *addr array*

Type: INTEGER Array.

Use: Elements of this array contain the numbers of the racks whose states are to be set by this call.

Format: INTEGER numbers in the range 0 to 10.

(3) *count*

Type: INTEGER Scalar.

Use: *count* specifies the number of racks, for which there are entries in *addr array*, whose states are to be set by this call.

8.1.3 Errors

STATE detects the fatal errors resulting either from calling it from the T/S part not in hybrid ready state, or from errors in the data—attempting to set a non-existent state or to set state in a non-existent rack.

8.2 Subroutine MODE (setting, addr array, count)

8.2.1 Purpose

Subroutine MODE is used for setting integrator and rack-common modes in the MAC.

8.2.2 Arguments

MODE requires three arguments:

(1) *setting*

Type: Single Precision Scalar.

Use: The caller loads this variable with a code specifying the particular mode setting being conveyed by this call.

Format: *setting* is specified as an ASCII string, of which only the leftmost character is interpreted as the initial letter of the mode being set ('I', 'R', or 'H', denoting INITIAL CONDITION, RUN or HOLD respectively).

(2) *addr array*

Type: Single Precision Array.

Use: Elements of this array are loaded with the addresses of the integrators or rack-common mode lines whose modes are to be set by this call.

Format: Addresses are specified in ASCII; integrator addresses take the form 'Armn', where r is the integrator's rack number ('0' to '9'), and mn is the integrator amplifier number ('10' to '19'). Rack-common mode settings are addressed with the form 'COMr', where r is the rack number ('0' to '9').

(3) *count*

Type: INTEGER Scalar.

Use: This variable is loaded with the number of integrators and rack-commons whose mode is to be set by the call.

8.2.3 Errors

MODE detects the fatal errors resulting either from calling it from the T/S part when not in hybrid ready state, or from data errors: attempting to set a non-existent mode, or attempting to set the mode of anything other than an integrator or rack common mode line.

As well, the software makes use of its internal data tables reflecting the MAC current mode and state settings to detect as fatal errors attempts to set modes incompatible with current states, for example trying to set an integrator to RUN when its rack is in STAND BY state. A MODE call may not address any computing element in a rack which is not OPCAL or COMPUTE state.

8.3 Subroutine TIMSCL (setting, addr array, count)

8.3.1 Purpose

Subroutine TIMSCL is used to set MAC integrator timescales. The default timescale setting is unity, so this subroutine is only needed when T/10 timescaling is required in the problem. Timescales are cleared by the DIGIBUS RESETs issued in response to HYBINI and HYBOFF calls.

TIMSCL cannot alter the time scale of integrators manually switched to the T/10 scale.

8.3.2 Arguments

Subroutine TIMSCL has three arguments, described below:

(1) *setting*

Type: Single Precision Scalar.

Use: *setting* specifies the timescale being conveyed to the MAC integrators by this call.

Format: A FORTRAN ASCII quantity, of which only the leftmost character is interpreted. Two values are allowed: 'R', denoting Real Time, and 'F' for Fast, or T/10 timescale.

(2) *addr array*

Type: INTEGER Array.

Use: Elements of this *addr array* are loaded with the numbers of the racks in which timescales are to be set by the call.

Format: INTEGER rack numbers, in the range 0 to 9.

(3) *count*

Type: INTEGER Scalar.

Use: Specifies the number of racks, for which *addr array* has entries, whose timescales are to be set.

8.3.3 Errors

TIMSCL detects the fatal errors resulting either from calling it from the T/S part when not in hybrid ready state, or from data errors.

9. ACCESS SYSTEM SUBPROGRAMS

Two subroutines allow the program to control the MAC access system address setting and to read the access system output, which is hardwired direct to one of the ADC input channels, through a 100 gain (limited) to another, and through a noise detection circuit (with nominal gain 10) to a third. An amplifier is automatically used by the access system when a DIFFERENTIAL CHECK (DIFF CHECK) access address is selected, and its gain (0·01, 0·1, 1·0 or 10·0) may be switched by the access address setting subroutine.

These subroutines may be used in the T/S part in hybrid ready state, or in the R/T part. When used in the T/S part, they invoke cycles of R/T execution internal to H3PAC to ensure their function is complete. A call from the T/S part to set the access system address will not return until the address has actually been transmitted to the access system. A call from the T/S part to read the access output will cause a new set of readings to be made, and will return these.

On the other hand in the R/T part the action of these subroutines is subjected to the timing of hybrid operation, as outlined in Section 2.1. An address set up by a call in a given R/T cycle will be transmitted to the access system after the start of the next R/T cycle, and a reading of the access system output at this address will not be available until during the R/T cycle after that.

The access system address and the DIFF CHECK amplifier scale setting are initialized to P000 (i.e. Analogue ground in rack zero) and 0 (gain 1·0) by the DIGIBUS RESET issued at HYBINI and HYBOFF. Otherwise they are only changed in response to the subroutine described below.

9.1 Subroutine ACCADR (class, rack, unit, gain)

9.1.1 Purpose

ACCADR is used to set the MAC access system access address and DIFF CHECK amplifier scale.

9.1.2 Arguments

There are four arguments to ACCADR:

(1) *class*

Type: Single Precision Scalar.

Use: This argument describes the type of device to be accessed.

Format: *class* is specified as a FORTRAN ASCII variable, of which only the leftmost character is interpreted. There are four possible values, 'A', 'D', 'P', and 'S', denoting Amplifier output, DIFF CHECK point, Power supply rail, and Special access input respectively.

(2) *rack*

Type: INTEGER Scalar.

Use: This argument specifies the rack number in the access address.

Format: INTEGER in the range 0 to 15.

(3) *unit*

Type: INTEGER Scalar.

Use: *unit* specifies unit number in the access address.

Format: An INTEGER, whose range depends on the setting of *class*:
(i) for *class* = 'A', the range is 0 to 39;
(ii) for *class* = 'D', the range is 0 to 19;
(iii) for *class* = 'P', the range is 0 to 9;
(iv) for *class* = 'S', the range is 0 to 29.

(4) *gain*

Type: INTEGER Scalar.

Use: This variable is used to control the gain of the DIFF CHECK amplifier, it is significant only when *class* = 'D', and may be omitted for other values of *class*.

Format: An INTEGER giving the common logarithm of the desired gain setting. It therefore has a range of -2 to $+1$.

9.1.3 Errors

ACCADR recognises fatal errors caused either by calling it from the T/S part when not in hybrid ready state, or by data errors: specification of an illegal address, or of an illegal gain when addressing a DIFF CHECK point.

9.2 Subroutine ACCVAL (value, value10, value1000, noise)

9.2.1 Purpose

ACCVAL returns the value of the currently accessed point in machine units in four variables, read at various hardware gain and ADC scale settings.

9.2.2 Arguments

ACCVAL returns four arguments:

(1) *value*

Type: REAL Scalar.

Use: This argument returns the value of the access system output in machine units.

(2) *value10*

Type: REAL Scalar.

Use: This argument returns the value of the access system output in machine units, read at an ADC scale setting of 10V nominal. This argument will more accurately represent the value of the access system output, provided that it is smaller than 0·1 m.u. in magnitude.

(3) *value1000*

Type: REAL Scalar.

Use: This argument returns the value in machine units of the access system output read at an ADC scale setting of 10V (nominal), through a gain of 100. This variable should be used when checking amplifier zeroes. It is significant only if smaller in magnitude than 0·001 m.u.

(4) *noise*

Type: REAL Scalar.

Use: This argument returns the average value of the peak-to-peak noise at the access system output, in machine units.

9.2.3 Errors

The only error detected by ACCVAL is the fatal error resulting if it is called from the T/S part when not in hybrid ready state.

10. MAC ERROR MONITORING SUBPROGRAMS

Errors in the MAC are sensed by the amplifier overload units on an amplifier-by-amplifier basis, and by amplifier oscillation detectors, servo trip indicators, and CAL and MULTIPLE CAL indicators in each of the racks. These are all available as DIGIBUS inputs, and are all read from the DIGIBUS by the PDP-11/20 software every R/T cycle.

The ADC operates at a real full scale of 105·0V (on the nominal 100V range), and has hardware overrange detection which may be individually enabled for each reading (see Section 5.1).

H3PAC has two functions which provide a quick indication of MAC error status plus a subroutine for detailed examination of the DIGIBUS error bits. They may be called from the T/S part when in hybrid ready state, or from the R/T part. Calling any of them from the T/S part when not in hybrid ready state will cause a fatal error.

When used in the R/T part, the MAC error indications returned are those read at the start of the R/T cycle in which the call is made (except that ERREAD returns a running logical sum of the error bits up to the current cycle, see Section 10.3). When called from the T/S part in hybrid ready state, a R/T cycle is completed and fresh data obtained by the subprograms before they return.

10.1 Logical Function MACERR (dummy)

10.1.1 Purpose

This function returns the value .TRUE. when there is at least one error bit set in the DIGIBUS input words, and .FALSE. when all are clear.

10.1.2 Arguments

One dummy argument of any type is required by the FORTRAN syntax for MACERR, it is ignored and unaffected by the call.

10.2 Logical Function ADCERR (dummy)

10.2.1 Purpose

This function takes on the value .TRUE. if any of the ADC input variables (for which overrange detection was enabled) were overrange, and .FALSE. if they were all within their appropriate ranges in the ADC scan made at the start of the R/T cycle in which the call was made, or, in the case of T/S part calls, in the scan made at the start of the asynchronous R/T cycle triggered at the start of the call.

10.2.2 Arguments

One dummy argument of any type is required by the FORTRAN syntax for ADCERR. It is ignored and unaffected by the call.

10.3 Subroutine ERREAD (bit array)

10.3.1 Purpose

ERREAD is used to obtain the complete set of MAC DIGIBUS input error indication words. The PDP-11/20 accumulates the logical sum (or 'running OR') of the words returned by each DIGIBUS input address each R/T cycle. These values are transferred to the DECsystem-10 when ERREAD is called, and the PDP-11/20 locations are cleared, so a call to ERREAD provides the history of the error status since the last time it was called, or since the HYBINI call, which also clears the locations. Thus, at T/S level, if MACERR returns the value .TRUE., ERREAD should be called twice, and the results of the second call used to locate which error bits are currently true.

10.3.2 Arguments

ERREAD returns one argument:

(1) *bit array*

Type: Single Precision Vector of dimension 16,

Use: The logical sum of the MAC DIGIBUS input words for the R/T cycles since the last call to ERREAD (or since HYBINI) are returned in DIGIBUS input format in successive halfwords of *bit array*.

Format: The format used is that of DIGIBUS inputs, packed two 16-bit words right-justified in successive left-half then right-half DECsystem-10 words. DIGIBUS input data formats are described in Systems Factors Group Memo 41, D/S 7. The order of the data is:

bit array	LH Contents		RH Contents	
element	Rack	DIGIBUS Adr	Rack	DIGIBUS Adr
1	0	48	0	49
2	0	50	1	48
3	1	49	1	50
.	.	..	.	..
.	.	..	.	..
.	.	..	.	..
.	.	..	.	..
15	9	49	9	50
16	10	48	10	49

11. DATA LOGGING SUBPROGRAMS

Some applications, for example MAC Console operation and hardware checkout programs, may not require to use the PDP-11/20's disks for data logging. Indeed the need to load and write-enable scratch disk packs could be a disadvantage in these circumstances. For this reason the user has the option, in the call to HYBINI (see Section 6.1 above), of initializing for logging or not. Where he elects not to log, the software does not concern itself with the status of the disks, and operation is consequently simplified. Of course calls to any of the data logging subprograms will give rise to error if the user has not indicated, in the HYBINI call, his intention to make such calls.

Data, consisting of the active analogue variables, the access system inputs, the DIGIBUS (MAC Error) input words and the ADC overrange flag (one word, cleared at the start of each set of ADC conversions and set to minus one if an overrange is detected during that set of conversions), are logged every R/T cycle.

The records from consecutive cycles are concatenated in one of two disk buffers, which accommodate 1536 words (six sectors), until the remaining space in the buffer is insufficient for the data from the next R/T cycle. At this time a transfer of the entire buffer to the disk is started, and the incoming data stream is directed to the other buffer. The maximum under-utilization of the disk capacity occurs when the records just exceed 1·5 sectors length, and is less than 25%.

Data on the disks is prefaced by a preamble record, starting at disk address zero, on drive zero, which is used for identification. The preamble contains a 100-character ASCII string, copied from the *ident* argument of HYBINI, a count of the number of active analogue variables, and a list of the ADC channels (multiplexer addresses) of the active variables. The preamble record is always six (256-word) disk sectors long.

The software allows overflow from disk drive zero to drive one if required during hybrid iteration. No preamble record is written on the second drive, and the information on it is altered only if an overflow on to it occurs. Both drives must, however, have disks loaded and be write-enabled before hybrid operation requiring logging can commence. This is checked by HYBINI (see Section 2.1 above). Logging ceases when disk 1 fills up.

Data records have a serial number, obtained by counting (the first record is number zero), which is accessible to the DECsystem-10 program to enable it to pass information to post-run data analysis programs enabling them to interpret the logged data.

When hybrid operation which involved logging is terminated by a call to subroutine HYBOFF, data acquired up to that time is output to disk, and the serial number of the last record written is written (in 16-bit unsigned integer form) as the first word in the last sector of the preamble record (sector 5, surface 0, cylinder 0, unit 0). This is also done (where possible) when hybrid operation terminates because of error.

Logging can be turned on and off by the DECsystem-10 program, or the current serial number obtained, using calls to the subprograms described below.

Many H3PAC subprograms, when called from the T/S part, need to invoke R/T cycles to ensure that their output is propagated before they return or that they use the most current values of input quantities. As has already been remarked, these R/T cycles are hidden from the user, in that they do not result in the execution of any user R/T code. Data is not logged for such cycles—it is logged only for those which involve actual execution of user R/T code, and only when enabled.

11.1 Subroutine LOGGO

11.1.1 Purpose

Subroutine LOGGO (which has no arguments) is used to start logging, or to cause logging to continue if it has been temporarily stopped. It may be called from the T/S part in hybrid ready state, in which case the logging will commence with the data from the first R/T cycle after ITERAT is called, or from the R/T part, in which case the data collected at the commencement of the cycle in which the call is made will be the first to be logged.

11.1.2 Errors

Fatal errors are detected if LOGGO is called and the HYBINI call had not specified that logging was to be done, or if LOGGO is called twice without an intervening call to LOGSTP (see Section 11.2) or if LOGGO is called from the T/S part in other than hybrid ready state.

11.2 Subroutine LOGSTP (serial)

11.2.1 Purpose

There may be periods during the execution of some hybrid computations when data logging is not required. LOGSTP provides a way of stopping logging, and if it is later decided that logging should resume, LOGGO will restart it. LOGSTP may be called from the T/S part in hybrid ready state, when the last set of data to be logged will be that obtained in the last R/T cycle executed before the job reverted from hybrid iterate to hybrid ready state; it would be futile to call LOGSTP from the T/S part if there had been no prior hybrid iteration activity. LOGSTP may also be called from the R/T part; it will cause logging to stop after the data collected at the beginning of the R/T cycle in which the call is made has been logged.

11.2.2 Arguments

LOGSTP returns one argument, the INTEGER scalar *serial*, which is the value of the serial number of the last data record logged before logging stops in response to the LOGSTP call.

11.2.3 Errors

LOGSTP detects fatal errors arising either from calling it from the T/S part when not in hybrid ready state, or from calling it when logging either has already been stopped or has not been started.

11.3 Integer Function LOGSER (dummy)

11.3.1 Purpose

To save the programmer the task of keeping a track of the serial number of the current log record, function LOGSER may be called either in the R/T part or in the T/S part in hybrid ready state to determine the serial number. When called in the R/T part, it returns the value of the serial number of the data recorded at the start of the R/T cycle in which the call is made. When called in the T/S part it returns the value of the serial number of the last log data record. If it is called before logging has been started with LOGGO, it returns the value -1.

11.3.2 Arguments

FORTRAN syntax requires that LOGSER have an argument—it is a dummy, and may be of any type. It is ignored and unaffected by the call.

11.3.3 Errors

A fatal error results if LOGSER is called from the T/S part when not in hybrid ready state.

11.4 Data Formats

All ADC data are recorded on the disks in the binary form presented at the output of the ADC.

11.4.1 Preamble Record

Consider the preamble record to be a singly-dimensioned array of PDP-11/20 words, with the index ranging from 0 to 1535. Words 0 to 49 inclusive contain the 100-character *ident* in PDP-11/20 ASCII form (characters contained in successive 8-bit bytes). Word 256 contains the count of active variables as a 16-bit integer, say n. Words 257 to $256+n$ contain the variables' multiplexer addresses as 16-bit integer channel numbers, with the multiplexer scaling and over-range control bits in them. Words $257+n$ to 1535 are zero, except that word 1280 will contain the *unsigned 16-bit integer* number of logging records actually written, on normal and most error terminations of hybrid operation.

11.4.2 Data Records

Logged data records are concatenated together in six-sector-sized (1536-word) buffers, and written sequentially on the disks. The order of the data within each record is the same. Words 0 to 31 contain the DIBIGUS input words, ten triplets from DIGIBUS unit addresses 48, 49 and 50 from racks 0 to 9, plus the pair from units 48 and 49 in rack 10. Word 32 contains zero if the ADC scan recorded no overrange reading, and -1 if there was an overrange reading. Words 33, 34, 35 and 36 contain the ADC output from the conversions of the access system output at, respectively, 100V (nominal) scale, 10V (nominal) scale (both direct), 10V (nominal) scale through 100 gain, and 10V (nominal) scale through the (gain 10) noise detector circuit. Subsequent words contain ADC data in the binary form presented at the ADC output.

12. CONNECTION OF PDP-11/20 FUNCTION GENERATORS

The operation of the PDP-11/20's digital function generators is described elsewhere by the author. Section 6.1 (above) explains the use of subroutine FGDATA for loading the function generator data tables.

The software allows up to ten digital function generators to be in operation at any time. Up to twenty data tables may be loaded, and any of the function generators may use any of the data tables. The action of patching an analogue function generator is replaced, in the case of the digital function generators, by the software connection of an ADC system input channel and a DCU (which must of course be patched to an appropriate source and amplifier) to one of the data tables. This can be done before hybrid iterations start, or from the user R/T code, allowing rapid switching of functions.

Function generators connected while the program is in hybrid ready state will be operated at the time of the call, and subsequently at any time that any of the H3PAC subprograms invoke a R/T cycle for data transfer. As well, when ITERAT is called, the digital function generators are operated at their sub-period rate for a full iteration period prior to the clock interrupt defining the start of the first R/T cycle. The aim of these provisions is to minimize start-up transients.

Overrange detection by the ADC is disabled for all function generator input variables.

12.1 Subroutine FGCONN (unit, output addr, table number, input addr)

12.1.1 Purpose

This subroutine is used to connect (or disconnect) a PDP-11/20 digital function generator to an input, an output, and a function data table. It may be called from the T/S part in hybrid ready state, or from the R/T part.

12.1.2 Arguments

Four arguments must be specified in the call to FGCONN:

(1) *unit*

Type: INTEGER Scalar.

Use: This argument defines the software unit number of the digital function generator being connected by this call. It must be an integer in the range 0 to 9.

(2) *output addr*

Type: INTEGER Scalar.

Use: The number of the DCU to which the output of the function generator is to be connected is specified by this argument.

Format: A DCU number is specified as a decimal INTEGER value of the form rmn, where r is the rack number (0 to 9), and mn is the DCU unit number (0 to 29).

(3) *table number*

Type: INTEGER Scalar.

Use: This argument specifies the function generator data table that is to be used by the function generator being connected by this call. The data tables are identified by the *table number* argument when they are loaded with FGDATA. Connecting a digital function generator to table number −1 causes it to be 'disconnected', the DCU to which it was connected prior to the call is unaffected.

Format: An INTEGER in the range 0 to 19.

(4) *input addr*

Type: Single Precision Scalar.

Use: This argument defines the analogue variable to be used as the input to the function generator being connected.

Format: The same format is used as for elements of the HYBINI argument *addr array* for defining active variables (Section 6.1). MAC Amplifier outputs are specified by an ASCII string of the form 'Armn', where r is the rack number ('0' to '9'), and mn is the amplifier number ('00' to '39'). ADC system patchable inputs are denoted 'Ilmn', where lmn is the ASCII representation of the input channel number.

12.1.3 Errors

FGCONN detects the fatal errors resulting from calling it from the T/S part when not in hybrid ready state. It also checks the validity of its arguments, for format and range, and notifies any errors as fatal.

13. USE OF MAC S-BUS AND IC/CAL-BUS

H3PAC has subprograms which enable the user to control the MAC S-bus relays, the S-bus DACs, and the IC/CAL-bus DAC. They may be called from the R/T part of the program, in which case their output to the hardware is done, in accordance with the timing described in Section 2.1, after the inputs for the next R/T cycle have been completed.

They may also be called from the T/S part in hybrid ready state, when they will cause the execution of R/T cycles hidden from the user to ensure that the hardware state is that specified in the calls when they return.

Three subroutines are used with the S-bus and IC/CAL-bus.

13.1 Subroutine SRELAY (value, addr array, count)

13.1.1 Purpose

As the name suggests, this subroutine is used for controlling the sixteen S-bus relays in each MAC rack. It may be called from the T/S part in hybrid ready state, or from the R/T part, see above for a description of operation timing in each of these cases.

The software keeps a record of the S-bus relay settings in the DECsystem-10, to enable it to pool its actions on a rack-by-rack basis during the execution of a single SRELAY call to save XIX transfers and to suppress redundant transfers.

S-bus relays are dropped when the DIGIBUS RESET is issued in response to the HYBINI call, and are thereafter picked and dropped only in response to calls to this subroutine until the HYBOFF call generates another DIGIBUS RESET which finally drops any still picked.

13.1.2 Arguments

Subroutine SRELAY requires three arguments:

(1) *value*

Type: LOGICAL Scalar.

Use: The value of this argument determines whether the relays specified are to be picked (*value* = .TRUE.) or dropped (*value* = .FALSE.).

(2) *addr array*

Type: INTEGER Array.

Use: Entries in this array are used to identify the S-bus relays to be picked or dropped.

Format: The S-bus relays are specified as integer numbers of the form rmn, where r is the rack in which the relay is located (0 to 9), and mn is the relay number in the rack (00 to 15).

(3) *count*

Type: INTEGER Scalar.

Use: This variable determines the number of S-relays, identified in *addr array*, to be picked (or dropped) by the call.

13.1.3 Errors

The software checks that the call to SRELAY has not been made from the T/S part when not in hybrid ready state, and checks the validity of the entries in *addr array*. Any errors detected are treated as fatal, the job stops and an appropriate error message is typed on the controlling terminal.

13.2 Subroutine SVALUE (addr array, data array, scale array, count)

13.2.1 Purpose

SVALUE is used for sending values to the S-bus DACs. It may be called from the T/S part in hybrid ready state, or from the R/T part, see above for a description of the timing of its operation in these cases.

13.2.2 Arguments

Four arguments are required by SVALUE:

(1) *addr array*

Type: INTEGER Array.

Use: Elements of this array specify the S-bus DAC numbers (INTEGERs in the range 0 to 15) to which values are to be sent as a result of this call.

(2) *data array*

Type: REAL Array.

Use: Elements of this array are loaded with the values to be sent to the DACs addressed by the entries in *addr array*. They are real numbers which when divided by their corresponding scales specified in *scale array* should not produce a result larger than 1·0 in magnitude.

(3) *scale array*

Type: REAL Array.

Use: Elements of this array are applied as scale factors to corresponding *data array* elements.

(4) *count*

Type: INTEGER Scalar.

Use: This argument tells the software how many S-bus DACs are to be loaded.

13.2.3 Errors

The software checks the legality of the call (it must not come from the T/S part when not in hybrid ready state), and the range of the arguments, reporting any errors as fatal.

13.3 Subroutine ICBVAL (value, scale)

13.3.1 Purpose

ICBVAL is used to send a value to the IC/CAL-bus DAC. It may be called from the T/S part in hybrid ready state or from the R/T part, see the description above of the timing of the execution of these calls.

The IC/CAL-bus is also used by subroutine INTINI, which may only be called from the T/S part in hybrid ready state. INTINI completes its operation by sending zero to the IC/CAL-bus DAC and will override any previous setting resulting from a call to ICBVAL. The IC/CAL-bus DAC is cleared at HYBINI and HYBOFF time.

13.3.2 Arguments

Two arguments, each a REAL scalar are used in the call to ICBVAL. The first, *value*, is used to specify the variable value, and the second, *scale*, its scale. Their quotient, which is the quantity actually present at the DAC output after completion of the call, must not exceed 1·0 in magnitude.

13.3.3 Errors

ICBVAL detects fatal errors resulting either from calling it from the T/S part when not in hybrid ready state or from the argument *value* being outside the range $-scale$ to $+scale$.

14. OPERATING ENVIRONMENT

Normal timesharing service on the DECsystem-10 is preserved while hybrid computations using H3PAC are carried out. The hybrid computation job's T/S part is scheduled for execution by the operating system in the same way as for all other jobs, although in a high priority run queue (HPQ 1). The R/T part computations are executed as interrupt code, and are not scheduled by the operating system. H3PAC automatically limits the total fraction of DECsystem-10 central processor (CPU) time used by the R/T part to 0·3. The programmer may, using subroutine CPFRAC (see below), elect to limit his job to a smaller fraction of CPU time consumption by the R/T part.

H3PAC accumulates run time statistics which may be examined using subroutine HSTATS. These two subroutines are described below.

14.1 Subroutine CPFRAC (setting)

14.1.1 Purpose

This subroutine allows the user to set the H3PAC limit on cumulative R/T part CPU time utilization to any value up to the 0·3 default limit.

CPFRAC may only be called when the job has normal timesharing status, that is before the HYBINI call.

14.1.2 Arguments

One argument is used by CPFRAC:

(1) *setting*

Type: REAL Scalar.

Use: *setting* is used to convey the new limit on cumulative R/T part CPU time utilization.

Format: A REAL Scalar.

Range: *setting* must lie in the range 0·0 to 0·3.

14.1.3 Errors

CPFRAC checks that it has been called before hybrid activity is initialized with HYBINI, and that the argument, *setting*, is in the correct range. Any errors are reported as fatal ones.

14.2 Subroutine HSTATS (cycles, total time, max time, min time)

14.2.1 Purpose

HSTATS, which may only be called from the T/S part, is used to interrogate H3PAC for the run time statistics which it collects during R/T execution. These quantities are cleared when hybrid operation is initialized with HYBINI, and are meaningless until hybrid operations are started with ITERAT.

14.2.2 Arguments

The four HSTATS arguments are:

(1) *cycles*

Type: INTEGER Scalar.

Use: This argument is used to return the total number of real time cycles executed in hybrid iterate state by the job since the HYBINI call which initialized real time activity.

Format: *cycles* is returned as an integer.

(2) *total time*

Type: REAL Scalar.

Use: This argument is used to return the (approximate) total DECsystem-10 CPU time used in the R/T part of the program since the HYBINI call which initialized hybrid operation.

Format: The quantity returned is a REAL number of seconds.

(3) *max time*

Type: REAL Scalar.

Use: This argument returns the maximum of the CPU execution times recorded for the R/T cycles executed since the HYBINI call.

Format: A REAL time in seconds.

(4) *min time*

Type: REAL Scalar.

Use: Returns the minimum of the DECsystem-10 CPU times recorded for the execution of the R/T cycles executed since the HYBINI call.

Format: A REAL time in seconds.

14.2.3 Errors

The only error detected by HSTATS is calling it from the R/T part. Omission of any of the arguments will result in program code corruption.

15. RUN-TIME SYSTEMS

H3PAC requires that the standard DECsystem-10 FORTRAN (F10) run-time system module FOROTS be loaded. This means that even if a program using H3PAC is composed entirely in the MACRO-10 assembler language, using, of course, the FORTRAN sub-program calling conventions, it must still be loaded with FOROTS. This requirement arises because of the way in which processor traps (for say floating over- or under-flow) are handled in the DECsystem-10 FORTRAN environment by the run-time system. FOROTS exists as a sharable section of code ('High Segment' in DECsystem-10 terminology), which is actually linked to the program only when it commences execution. It locks in core with the H3PAC program 'Low Segment' at HYBINI time. Probably the simplest way to ensure that any program using H3PAC actually does establish the run time connection with FOROTS is to make the outer-most program module a FORTRAN program—it may do no more than merely call the actual operational code.

Programs using H3PAC must be loaded with the system software module LOKXIX. This has two main functions: the first is load checking, and the second is management of the XIX interface. The load checking function ensures that the object (relocatable binary, or '.REL' in DECsystem-10 terms) modules making up a particular program are loaded in the order: first (nearest logical address zero) LOKXIX, then H3PAC and any other modules which use LOKXIX (for example CISPAC, the CIS Display driver package), and last all user program modules. This loading order allows the validity of addresses for data to be written by H3PAC to be checked, reducing the probability that system-critical code, such as the XIX interrupt handler, will be corrupted should a wrong address be generated by a user program module.

Code for XIX interface management is in LOKXIX so that it can be accessed by the other software packages which may need to use the XIX for communication with the PDP-11/20. These include CISPAC, for CIS Display use, and RKPAC, the package of FORTRAN-callable routines for using the PDP-11/20's disks (described in the next chapter). Putting the code in one place allows it to be shared by these modules, saving core and simplifying the management of their activities should these be concurrent. It is not possible for more than one single job to make use of the XIX interface for communication with the PDP-11/20 at any one time, and LOKXIX also enforces this restriction.

If the program modules are not loaded in the correct order, a message to this effect will be output on the job's controlling terminal when an attempt is made to execute the loaded program, and the execution will not proceed.

The next chapter discusses RKPAC, and explains that while it may be loaded in the same program as H3PAC, the two packages may not be used together. It is possible for CISPAC and H3PAC to be used at the same time: certain of the CISPAC routines may be called from the real time part of a hybrid program. The CISPAC manual provides a guide to the use of CISPAC routines in conjunction with H3PAC.

16. RETURN OF LOGGED DATA

The data logged on the PDP-11/20's disks during hybrid computation may be read back into the DECsystem-10 for analysis using calls to subprograms in the software package RKPAC. However this cannot be done during hybrid activity, because H3PAC and RKPAC both need to control the XIX interface. Calls to RKPAC subprograms made after a call to HYBINI but before a call to HYBOFF are considered fatal errors, the job stops and an appropriate error message is output on the controlling terminal.

The user requires the same privilege bits to use RKPAC as are required for H3PAC. Use of any of the RKPAC subprograms by an unprivileged job is a fatal error, the job stops and an appropriate error message is typed on the controlling terminal.

RKPAC may be loaded with the hybrid program, or separately in a post-run analysis program. The RKPAC subprograms are described below.

16.1 Subroutine RKSEEK (unit, cylinder, surface, sector, error)

16.1.1 Purpose

RKSEEK is used to position the heads prior to performing data transfers from (or to, see Section 16.3, below) the PDP-11/20's RK05 disks. Refer to the PDP-11/20 Peripherals and Interfacing Handbook for more detail.

16.1.2 Arguments

RKSEEK requires four input arguments, specifying the positioning command, and returns a fifth argument indicating whether the command has successfully been executed and an error message in case of failure. The arguments are:

(1) *unit*

Type: INTEGER Scalar.

Use: This argument specifies which disk pack drive's heads are to be positioned by the call.

Format: An INTEGER, either 0 or 1.

(2) *cylinder*

Type: INTEGER Scalar.

Use: This argument specifies the RK05 cylinder address to which the heads are to be moved.

Format: An INTEGER, in the range 0 to 202 (Decimal).

(3) *surface*

Type: INTEGER Scalar.

Use: This argument specifies the RK05 surface to be selected.

Format: An INTEGER, having the value 0 (for the upper surface), or 1 (for the lower).

(4) *sector*

Type: INTEGER Scalar.

Use: This argument specifies the sector address to be selected.

Format: An INTEGER, in the range 0 to 11 (Decimal).

(5) *error*

Type: INTEGER Scalar.

Use: The software returns error codes, in the form of negative integers, in this argument, and zero if the operation was carried out successfully. The argument can therefore be tested as a LOGICAL variable, which takes the value .TRUE. if an error was detected, and .FALSE. if the seek operation was successful.

Format: The error codes returned by the subprograms of RKPAC are listed in Section 16.4.

16.2 Subroutine RKREAD (data array, count, error)

16.2.1 Purpose

RKREAD is used to read a block of data from one of the PDP-11/20's RK05 disks into an array in the DECsystem-10's core. Up to 3072 PDP-11/20 words of 16 bits packed right justified and sign extended in the successive 18-bit left- and right-halves of (up to 1536) DECsystem-10 words may be read, this is one full track of twelve 256-word sectors on the disks.

The drive and disk address from which the data originate is determined initially by a call to the subroutine RKSEEK (see above), which must precede any sequence of calls to RKREAD. On successful return from RKREAD, the disk address is advanced to the sector following that from which the last data word transferred came. To read contiguous data from a drive it is therefore only necessary to perform one RKSEEK call, followed by RKREAD calls as required.

After an RKREAD call in which any hard (drive or controller function) error was detected, the position of the heads is no longer defined, and a new call to RKSEEK must be made before attempting to read data again. In the case of a soft (data) error, the heads are returned to their position prior to the commencement of the call in which the error occurred, allowing retry of the operation if desired.

16.2.2 Arguments

RKREAD has three arguments:

(1) *data array*

Type: Single Precision Array.

Use: Data from the disks is deposited sequentially, right justified and sign-extended to 18 bits in left- and right-halves of elements of *data array*. If an odd number of PDP-11/20 words is transferred, the right half of the last DECsystem-10 word will be undefined. The user is cautioned that the contents of *data array* may be changed by the execution of a call to RKREAD in which an error is detected and from which *error* is returned with a non-zero value.

(2) *count*

Type: INTEGER Scalar.

Use: This argument is used to specify the number of *PDP-11/20 words* to be transferred . from the selected RK05 into *data array* by this call.

Format: An INTEGER, in the range 0 to 3072 (a call specifying zero words does not do anything).

Range: *count* can have any value from 0 to 3072, provided it does not cause the disk to attempt transfer data from a cylinder address greater than 202 (decimal).

(3) *error*

Type: INTEGER Scalar.

Use: This argument is cleared by the software if the data transfer invoked by RKREAD was performed successfully, and set to a negative error code if an error was detected. It can therefore be tested as a LOGICAL variable, which takes the value .TRUE. if an error occurred, and .FALSE. if the transfer was executed correctly.

Format: The error codes returned by the subprograms of RKPAC are listed in Section 16.4.

16.3 Subroutine RKWRIT (data array, count, error)

16.3.1 Purpose

This subroutine, which permits the DECsystem-10 programmer to write data on the PDP-11/20's disks, is not required for hybrid computation. Its description is included here for the sake of completeness.

RKWRIT is used to write blocks of data from an array in the DECsystem-10's core on to one of the PDP-11/20's RK05 disks. Up to 3072 PDP-11/20 words, obtained as the low-order 16 bits of the left- and right-halves of (up to 1536) successive DECsystem-10 words can be written with a call to RKWRIT. The drive and disk address to which the data goes is specified initially by a call to RKSEEK, which must precede any sequence of calls to RKWRIT.

On successful completion of a call to RKWRIT, the last disk sector written in is zero filled, and the disk address advanced to point to the next sector. Contiguous data on disk can therefore be produced with a single call to RKSEEK followed by a series of calls to RKWRIT.

After a call in which an error was detected, the current head position is not defined, and RKSEEK must be called before attempting any further data transfers. Also, in case of error the data written on the disk is undefined.

16.3.2 Arguments

RKWRIT has three arguments:

(1) *data array*

Type: Single Precision Array.

Use: Data is copied from this array on to the disks, successive PDP-11/20 words are are formed from the low-order 16 bits of the left- and right-halves of the entries in *data array*.

(2) *count*

Type: INTEGER Scalar.

Use: *count* specifies the number of *PDP-11/20 words* to be written on the disk by this call.

Format: An INTEGER, in the range 0 to 3072.

Range: *count* must not be such that it causes the drive to attempt to write on a cylinder whose address is greater than 202 (decimal).

(3) *error*

Type: INTEGER Scalar.

Use: RKPAC sets this argument to zero before returning from a successful call to RKWRIT, and to a negative error code on return from a call during the execution of which an error was detected. It can therefore be tested as a LOGICAL variable, as explained above.

Format: The error codes returned by Subroutine RKWRIT are listed in Section 16.4.

16.4 RKPAC Error Codes

The three RKPAC subprograms described above always return control to the caller, except in the case where one of them is called during hybrid activity, when the job stops and an appropriate error message is output to the job's terminal. They all have an *error* argument, which is set to zero on return from successful calls, and which contains an error code, in the form of a negative integer, on return from a call in which an error was detected. Since the RKREAD and RKWRIT operations can involve disk seeks (refer to the PDP-11 Peripherals and Interfacing Handbook), the error codes detected make up a set common to three subroutines.

The possible values of the *error* argument are:

0 — Denotes a successful call.

−1 — Indicates PDP-11/20 power down.

−2 — Indicates XIX interface not available, because some other job is using it.

−3 — Indicates that the PDP-11/20 is not responding or is responding incorrectly to the DECsystem-10.

−4 — Indicates that an attempt has been made to seek an address on a non-existent disk drive.

−5 — Indicates that an attempt has been made to seek an illegal cylinder address, or that the current head position is such that execution of the requested data transfer will result in an attempt to transfer data to or from a cylinder whose address is greater than 202 (decimal).

−6 — Indicates that an attempt has been made to seek an illegal surface.

−7 — Indicates that an attempt has been made to seek an illegal sector.

−8 — Indicates that the drive selected is indicating power low.

−9 — Indicates that the selected drive is unready.

−10 — Indicates operation failed due to hardware malfunction.

−11 — Indicates RKREAD or RKWRIT call not preceded by RKSEEK.

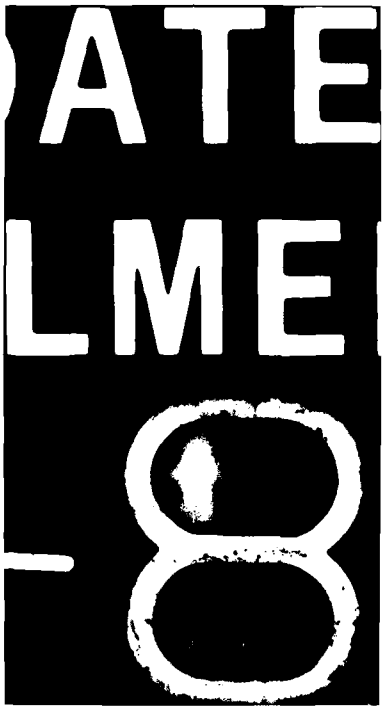
−12 — Indicates RKREAD or RKWRIT argument *count* is illegal.

−13 — Indicates that a soft error occurred during the execution of the call.

−14 — Indicates that an attempt has been made to write on a drive which is write-protected.

DISTRIBUTION

	Copy No.
AUSTRALIA	
Department of Defence	
Central Office	
Chief Defence Scientist	1
Deputy Chief Defence Scientist	2
Superintendent, Science and Technology Programmes	3
Australian Defence Scientific and Technical Representative (UK)	—
Counsellor, Defence Science (USA)	—
Joint Intelligence Organisation	4
Defence Central Library	5
Document Exchange Centre, D.I.S.B.	6–22
DGAD (NCO)	23
Aeronautical Research Laboratories	
Chief Superintendent	24
Library	25
Superintendent—Systems Division	26
Divisional File—Systems	27
P.O. Flight Systems Group	28–33
Author: H. Thelander	34
Materials Research Laboratories	
Library	35
Defence Research Centre, Salisbury	
Library	36
Department of Industry and Commerce	
Government Aircraft Factories	
Library	37
D. G. Bryant	38
Statutory, State Authorities and Industry	
CSIRO, Division of Computing Research	39
UNITED KINGDOM	
Royal Aircraft Establishment, Farnborough, Dr. A. A. Callaway	40
Universities and Colleges	
Manchester Library, U.M.I.S.T.	41
UNITED STATES OF AMERICA	
NASA Scientific and Technical Information Facility	42
Battelle Memorial Institute, Library	43
Universities and Colleges	
Johns Hopkins Library, Applied Physics Laboratory	44
Spares	45–54


ATE
LME
8